

PROTOMOL: Object-oriented framework for Prototyping Novel Algorithms for Molecular Dynamics

Author: Matthey *et al.*

Presenter: Santanu Chatterjee

Computational Biology
(CSE598K)

Fall 2004

PROTOMOL design issues

- Rapid development of efficient algorithms and applications in molecular modeling.
- Existing programs in this area are efficient but complex and intimidating for new developers.
- Optimized algorithmic framework along with components to handle I/O, GUI and visualization for easy incorporation and testing of new algorithms.
- In essence, to realize both efficiency as well as flexibility.

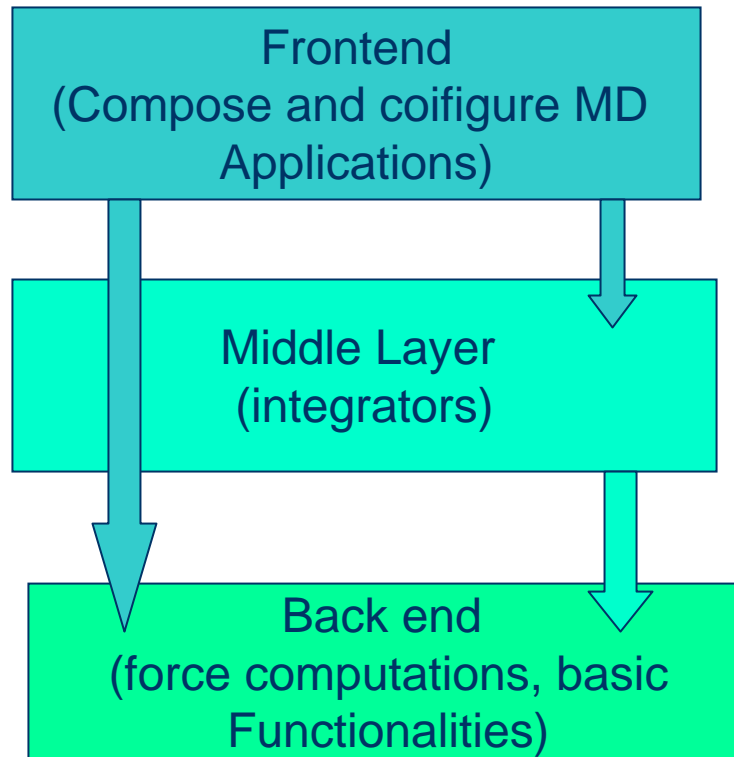
Challenges

- Different force types with different time scales of dynamics. Incorporation of both small (STS) and large (MTS) time stepping integrators.
- Different types of possible boundary conditions: Vacuum, PBC.
- Optimization of Non-bonded interactions: cut-off for pair-wise interactions, switching functions.

Goals of framework design

- Let the end-user experiment with different integration schemes. Composition of MTS integrators dynamically.
- Easy integration option for novel force algorithms.
- Abstraction of parallelization approach.
- Comparison of accuracy, stability, run-time efficiency etc. for MD algorithms.

PROTOMOL design



PROTOMOL Front-end

- Performs pre (I/O facilities to setup initial state) and post processing (output).
- Setting up Simulation object with correct topology format, forces, energies.
- Setting up desired integrator scheme from the configuration file.

Dynamic definition of integrators

- Integrator definition language (IDL).
- Allowing multi-level integrator with level 0 always a STS integrator for high-frequency force terms.
- Integrator class hierarchy to support IDL. Designed with inheritance. A `run()` method at the abstract integrator base class implements specific scheme.
- Abstraction between MTS and STS:
doDriftOrNextIntegrator(): executes next level of integrator for MTS, position update for STS.

Force computation

- Requirements to compute non-bonded interactions:
Algorithms to select n-tuples, boundary conditions, Cell managers, kernel, switching functions.
- Designed with templates and inheritance to decouple first step from the rest.
- JIT compiler to map user-definition into force object.

Parallelization

- Methods *evaluate()* and *parallelEvaluate()* to evaluate force computations. The former is a pure virtual function.
- Based on force decomposition with master-slave or static work allocation.
- Moderate scalability.

Performance monitoring

- Allows comparison of different MD algorithms.
- Efficient implementation of algorithms by encapsulating commonalities using inheritance.
- On-the-fly comparison to reduce overhead in current simulation.

NAMD2: greater scalability for Parallel Molecular Dynamics

Author: Kale *et al.*

Presenter: Santanu Chatterjee

Computational Biology
(CSE598K)

Fall 2004

Motivation

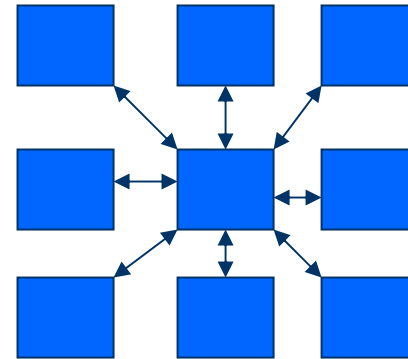
- Parallel computing provides the potential to address the complexity of large-scale MD simulation.
- Scalability is crucial for parallel implementations for efficient utilization of parallel architecture.
- Novel decomposition strategy is required for non-bonded force computations without violating bonded interaction principles.

Parallelization approaches

- Replicate Data: communication cost is $P \log P$, non scalable.
- Atom decomposition: arbitrary decomposition of atoms, $O(N)$ communication, not scalable.
- Force decomposition: distributes force matrix into different processors, not Isoefficiently scalable.
- Spatial decomposition: assigning neighboring atoms to same processor. Load imbalance for non-periodic systems.

NAMD2: Object-oriented design

- Based on class hierarchy.
- Core classes:
 - Patch
 - Compute
 - Sequencer.
- Multiple patches in one node.



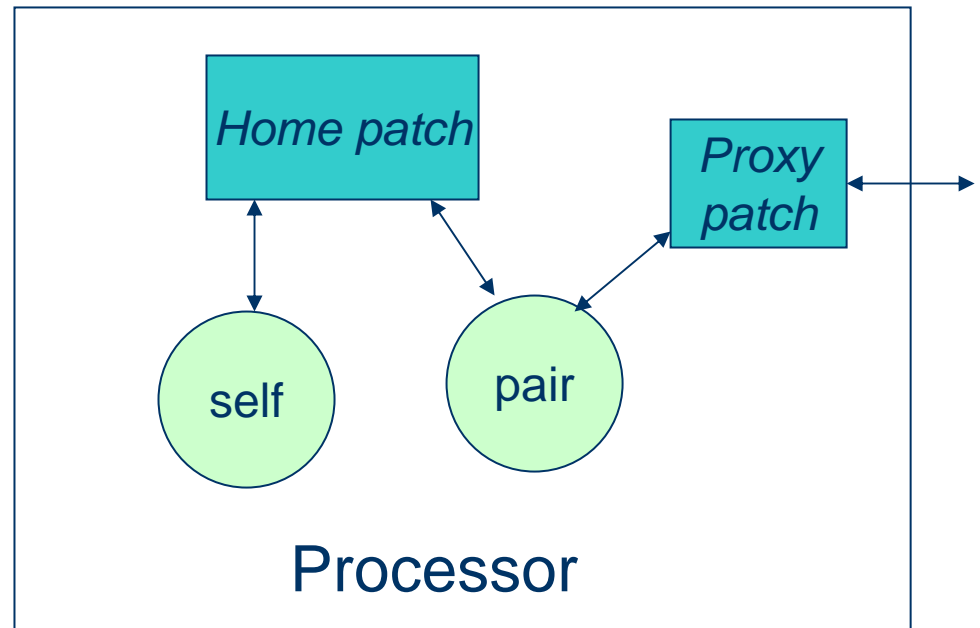
NAMD2: Object-Oriented Design

- Patch class implements spatial decomposition with dimension slightly larger than non-bonded cutoff.
- Compute class: force computation, depends upon multiple patches.
- Proxy patch optimizes the dependency of compute and patch objects by minimizing communication.
- Sequencer: thread associated with home patch and provides decision making type of capabilities.

NAMD2: Force computation algorithms

- Size of patch: *cutoff+margin*.
- Non-bonded force computation: self and pair compute object.
- Self: calculates forces for pair of atoms in same patch.
- Pair: calculates forces between pairs of atoms in neighboring patch. Depends upon proxy-patch.

NAMD2: Force computation algorithms



NAMD2: Force computation algorithms

- Bonded interactions: single compute object for each type of bonded interactions.
- Calculation for the pair of atoms at the common downstream patch. This results in reduction in amount of messages.
- Same proxies required for both bonded and non-bonded force computations.

NAMD2: Load balancing

- Initial load balancing during program startup.
- Patch distribution with evenly divided atoms at each node.
- Self compute objects at the same node with the patches.
- Pair compute object is distributed to the upstream neighbors.

NAMD2: Dynamic load balancing

- Min-heap stores current load in each processor. Lightly loaded processor on top of the heap.
- Migratable objects are on a max-heap, along with the proxies on which they depend.
- Migratable object is assigned to a new processor if the modified load is less than average.